Lab: Maximum Sum Contiguous Subsequence

Problem: In computer science, the maximum subarray problem is the task of finding the contiguous subarray within a one-dimensional array of numbers (the array contains at least one positive number) which has the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the contiguous subarray with the largest sum is 4, −1, 2, 1, with sum 6.

The problem was first posed by Ulf Grenander of Brown University in 1977, as a simplified model for maximum likelihood estimation of patterns in digitized images. A linear time algorithm was found soon afterwards by Jay Kadane of Carnegie-Mellon University (Bentley 1984). (http://en.wikipedia.org/wiki/Maximum_subarray_problem)

Lab, Part 1: Designing an algorithm (follows Bentley - Programming Pearls)

To start your goal is **not** to develop the most efficient algorithm. Rather it is **to apply the design paradigms you have been learning and develop different solutions**. At the same time you will reinforce your understanding of asymptotic analysis of algorithms by deriving the Θ() bounds for each approach. **As you work through the following, use the attached template to record your results in a document, clearly labeling each improvement and your derivation of its computational complexity.** In your derivations indicate the line/loop in the code that justifies each term in the summation.

The **basic operation is addition**.

Brute force Solutions

Consider a brute force solution to this problem A[1..N] is an array containing a sequence of integers

Initial solution

```
1.   int maxSoFar = 0;
2.   for low = 1 to N
3.        for high =  low to N
4.             int ThisSum = 0;
5.             for k = low to high
6.                  ThisSum = ThisSum + A[ k ];
7.             if( ThisSum > maxSoFar )
8.                  maxSoFar = ThisSum;
9.   return maxSoFar;
```

A. **1) Derive the asymptotic computational complexity of this solution. Hint: Work inside out. See Appendix A in Levitin**

Improvement 1: By removing redundant calculations find an improvement that gives a quadratic solution

B. **Quadratic Solution**
   1) Clearly write the pseudo code (using notation above as much as possible) of your algorithm.
   2) Show that your solution is quadratic by deriving the asymptotic computational complexity of this solution.

Improvement 2

C. Design an n * log n Divide and Conquer Solution

1) Clearly write the pseudo code of your algorithm
2) What is the recurrence relation for the divide and conquer solution?
3) Derive the asymptotic computational complexity of this solution?

Improvement 3:

D. Design a Scanning solution

Try to develop a solution that scans the numbers linearly and keeps track of certain values, so that the best solution can be found.

1) Clearly write the pseudo code of your algorithm
2) Derive the asymptotic computational complexity of this solution?

Lab, Part 2: Design and implement an experiment to test empirical results against the predicted asymptotic results

E. Empirical and Theoretical Results compared .

Read the program provided, it will be uploaded next class. It implements the four solutions discussed in part 1. Develop a hypothesis of algorithms' performance on arrays of various sizes.

1) What curves would you expect to find that would fit the data?

Run the program for different sizes of the array to determine what the appropriate sizes for the experiment should be. Make sure you understand what the duration numbers mean. To reduce the variability in duration from run to run, you should stop as many other programs as possible that may use resources concurrently with your runs.

Also, you will want to do multiple runs for each array size and take their average. (The amount of time a run for a fixed array size will vary due to the other processes running on the machine.) Finally, you will need to use different sizes of arrays for the different algorithms to be able to see the shape of the curves.

Once you have a set of array sizes to test for each algorithm. Run the tests. Record the results in you write up. Make sure to indicate the number of runs for a given array size. You may recall from statistics that more runs reduce the variability of you estimate of the mean (average time) that the algorithm takes for a given size array on your computer.

2) Graph the results to compare your results to the asymptotic predictions.
3) Do the empirical results match the theoretical curves? Why or why not?

A. 1. Derive the asymptotic computational complexity of this solution?

B. Quadratic Solution

1. Clearly write the pseudo code of your algorithm

2. Derive the asymptotic computational complexity of this solution?

C. Design an n * log n Divide and Conquer Solution

1. Clearly write the pseudo code of your algorithm

2. What is the recurrence relation for the divide and conquer solution?

3. Derive the asymptotic computational complexity of this solution?

D. Design a Scanning solution

1. Clearly write the pseudo code of your algorithm.

2. Derive the asymptotic computational complexity of this solution?

E. Empirical Exploration

1. What curve would you expect to find that would fit the data for each of the algorithms?

2. Graph the results to compare your results to the asymptotic predictions.

3. Do the empirical results match the theoretical curves? Why or why not?